XPRESS Project

PI: Ron Brightwell, Sandia National Laboratories

Executive Summary

The XPRESS (e**X**ascale **PR**ogramming **E**nvironment and **S**ystem **S**oftware) Project is developing the software stack for 21st Century extreme scale computing. It is sponsored by the DOE Office of Science ASCR X-Stack Program, and is beginning its second year of intense research and development combining the resources and talents of Sandia National Laboratories, Oak Ridge National Laboratory, Indiana University, University of Oregon, Louisiana State University, University of North Carolina-RENCI, Lawrence Berkeley National Laboratory, and Oak Ridge National Laboratory. The XPRESS project is forwarding the goals of this evolving exascale initiative through on-time accomplishment of its principal milestones, new discoveries related to the operation and structures of future dynamic adaptive software systems for DOE mission-critical applications and leadership class execution platforms, and demonstrating early prototypes of component layers that will comprise the exascale computers of the next decade.

XPRESS is unique among the X-Stack projects in that it addresses 1) the full software stack, 2) and is guided by a holistic parallel execution model for efficiency and scalability. It is the only X-stack project that is developing a new operating system expressly for this performance domain and the custom interface (RIOS) between this lightweight kernel OS (LXK) and its co-designed runtime system software (HPX). LXK is order constant scaling, eliminates OS noise, and extends world-class research in lightweight kernel supervisors including the commercially released Catamount OS. HPX-4 exploits early work on the shared memory multi-threaded message-driven HPX-3 experimental runtime system incorporating significant semantic advances and efficiency mechanisms. One unique property of the XPRESS system is its innovation of multi-nodal processes. Another is its introspective RCR black-boarding of system state and its APEX application instrumentation and runtime control. An experimental low-level programming model, XPI, is being provided for early application-driven experimentation and evaluation. Another unique contribution of the XPRESS research project is the enhancement and refinement of the ParalleX execution model to guide the co-design of the system software component layers and govern their interoperability.

Consistent with and governed by its program plan, XPRESS has had dramatic success in advancing the state of the art towards the realization of effective and comprehensive exascale system software stack. This complex collaboration of

half a dozen cooperating institutions has been organized with multiple PI meetings, a number of technical focused topic meetings, and bi-weekly telecons. Progress has been achieved with the experiment to interconnect a dynamic runtime system, HPX-3, with a lightweight kernel OS, Kitten, to derive critical experience with this breakthrough class of system software. This has contributed to the early draft of the RIOS specification of an interface between the application side and machine side software components. From the user side, the specification of the new low-level parallel programming interface, XPI, has been completed and is in review in preparation for formal release. This experimental library will enable experimentation with advanced parallel algorithms towards a new generation of scalable scientific applications. Software elements of the HPX-4 runtime systems have been developed, tested, and evaluated including the threads package, multiple forms of the parcels communication package, early forms of global address spaces, and local control object synchronization. Parallel distributed execution with proxy apps including GTC and LULESH as well as AMR codes have been implemented and executed on multicore and multi-node parallel systems, a key milestone for the XPRESS project and proof of concept. The software architectures of the ephemeral APEX instrumentation and control component as well as the persistent RCR black-boarding component have been completed in initial form. A completely new version of the ParalleX Report (5.2) has been released incorporating important refinement to this overarching execution model. At the recommendation of ASCR leadership, the SLOW performance model has been extended to incorporate parameter associated with energy and reliability to form the new SLOWER model. As planned, the applications-focused research work under the leadership of Mike Heroux has just been initiated. The XPRESS project collaborates, contributes to, and benefits from the collaboration with a number of other research projects within DOE and other agencies including NSF, DOD, and NSA. Finally, a recent unintentioned discovery is the property of emergent behavior through the synthesis of the LXK operating system and the HPX runtime system to establish a symbiotic system-wide protected supervisor for total system resource management and coordination. This new derived layer, PRIDE (Parallel Resource Integration for Distribute Execution), is a natural consequence of the runtime and OS capabilities and integration, and will require a minimum amount of additional coding. (This new direction, although promising, is not funded and has not been included in the current project plan.)

Immediate future work is proceeding aggressively under the project plan and Sandia's direction. Most critically are the software integration of the subcomponents of the runtime system to implement the HPX-4 runtime system software and the further interface of this with the LXK node operating system to achieve a first realization of the XPRESS software stack. The XPI programming library is under development with first implementation to be deployed within two months. More Co-design Center proxy apps will be developed for XPRESS including Boxlib AMR and Neckbone, among others. The Portals 4 networking layer will be integrated with LXK and providing services to HPX. In cooperation with the ASCR Execution Models projects, two different formal specifications of ParalleX will be developed using both operational semantics and an abstract state machine (ASM) formalism. The introspection layer comprising the integration of APEX and RCR will be implemented for dynamic control. Together, these activities and anticipated accomplishments will prepare XPRESS for its series of in-depth experiments and evaluation compliant with the project planned milestones.

Year 2 Progress

HPX-3

LSU has continued to update, develop, and improve HPX. Several key enhancements have been added including support for the BlueGene/Q and Xeon Phi, object migration, and refactoring of the threading sub-system and parcel port. LSU has implemented the basic framework for object migration that allows users to utilize this tool in their code. This concept will become more centerpiece as we begin to incorporate runtime information into HPX's decision-making processes. The threading system in HPX-3 has been overhauled and cleaned up significantly to improve overall performance of the threading and scheduling subsystem, and to improve the extensibility and maintainability of the code base to simplify adding new schedulers while being able to reuse large parts of the existing code base. The parcel port has been refactored so that users can easily target the communication layer of their choice. These changes are greatly improving the portability and efficiency of the HPX code.

LSU is also developing a proof-of-concept implementation of XPI on top of HPX-3. This implementation, HPXPI, is nearing an initial release. This release will allow the XPRESS team to write and test code utilizing the XPI interface.

HPX-4

The XPRESS team has defined the major components of the HPX-4 runtime system as well as the interfaces between major system components. There are working implementations of the parcel handling, LCOs, and networking components. IU is actively working with LSU and UO/RENCI to integrate the threads and performance components, respectively. Although the global address space component has yet to be defined, there are several successful distributed runs with the HPX-4 runtime on up to 512 cores with a port of the LULESH application.

The parcel handler in HPX-4 is capable of triggering futures on a remote node as well as performing large message transfers between nodes. The LCO component has implemented futures and gates (which are being used by the

LULESH port), and IU is currently working on defining the semantics for dataflow LCOs. IU and LSU have come to an agreement that the interface between the threads package and LCOs is sufficient to handle expected use cases. In addition, the networking in HPX-4 is currently supported by Photon (an IU project). IU is actively working towards supporting Portals 4, particularly for its RDMA capabilities.

XPI

The XPI specification developed by IU has reached Beta status. It contains a large number of changes from its initial version six months ago. In particular, 1) an interface for attaching parcels to processes was determined necessary to fully interface with termination detection, 2) an interface for testing to see if a parcel's wait queue is empty has been added to streamline synchronization, and 3) an interface to allocate and inspect process global data has been added. In addition, a number of small changes and bug fixes have been included in the C-interface specification as we gain experience writing algorithms with XPI. Finally, a number of small examples have been included directly in the document, and some larger examples have been developed externally. The beta specification is being implemented concurrently by IU and LSU with expected initial, feature releases this spring. When these releases become available, the XPI specification will move out of beta and become XPI version 1.

APEX

The University of Oregon (UO) has begun work on creating a new performance data API for evaluating metrics mapped from low-level measurements during execution to allow for performance data introspection. In order for APEX to observe the performance state of an XPRESS application running in the OpenX stack, each layer of the stack requires access to the APEX component. That access will be provided through an application programming interface (API) for other components to both register events with the APEX component, as well as request notification when performance conditions are such that intervention is necessary. The intervention will be managed by an APEX subcomponent called the Policy Engine. Modeled after the initial integration with HPX-3, a more expansive APEX API has been drafted, and is currently under review with the other members of the XPRESS team. In addition, we are working to help define the API between APEX and the RCRBlackboard, in order to provide introspection into the hardware and operating system layers. RCR is critical to providing the full system view, and APEX will provide the conduit for higher layers of the software stack – including the application layer – to the operating and runtime system state.

Second, we have begun refactoring the initial APEX prototype, in order to align with the requirements of the Policy Engine and to target HPX-4. Previously, the APEX component served as a proxy for TAU profile collection, but as the requirements for APEX have evolved so has its prototype implementation. APEX has been redesigned to function in two modes: primarily as an event-driven architecture with a secondary periodic interrogation component. At initialization, listener components register with APEX to be notified when events happen in the system. The TAU integration has been refactored to be one such component, and can be optionally disabled at either configuration time or at runtime. When appropriate, each HPX thread reports its change in state to APEX. When an event occurs, each of the registered listeners is notified of the event through the registered callbacks. Optionally, components can periodically interrogate the runtime state and perform whatever analysis they require. In addition to the TAU profile collection component, a concurrency component has also been prototyped using both the event-driven architecture and the periodic interrogation. As described in previous reports, APEX has reverse engineered the Intel Instrumentation and Tracing Technology (ITT) API, because the HPX-3 thread scheduler has already been instrumented with ITT API calls. The thread scheduler in HPX-3 is using the ITT interface to report when tasks are scheduled on operating system threads. The concurrency listener keeps track of the current task for each thread of execution. In addition, the concurrency component periodically interrogates the task state for the threads, and generates a timeline activity graph showing how many threads are active during each period. Figure 1 shows some sample output from the concurrency component. The timeline shows how many threads were executing and in which functions during each sampled event. Figure 2 shows the TAU profile collected during the same execution.

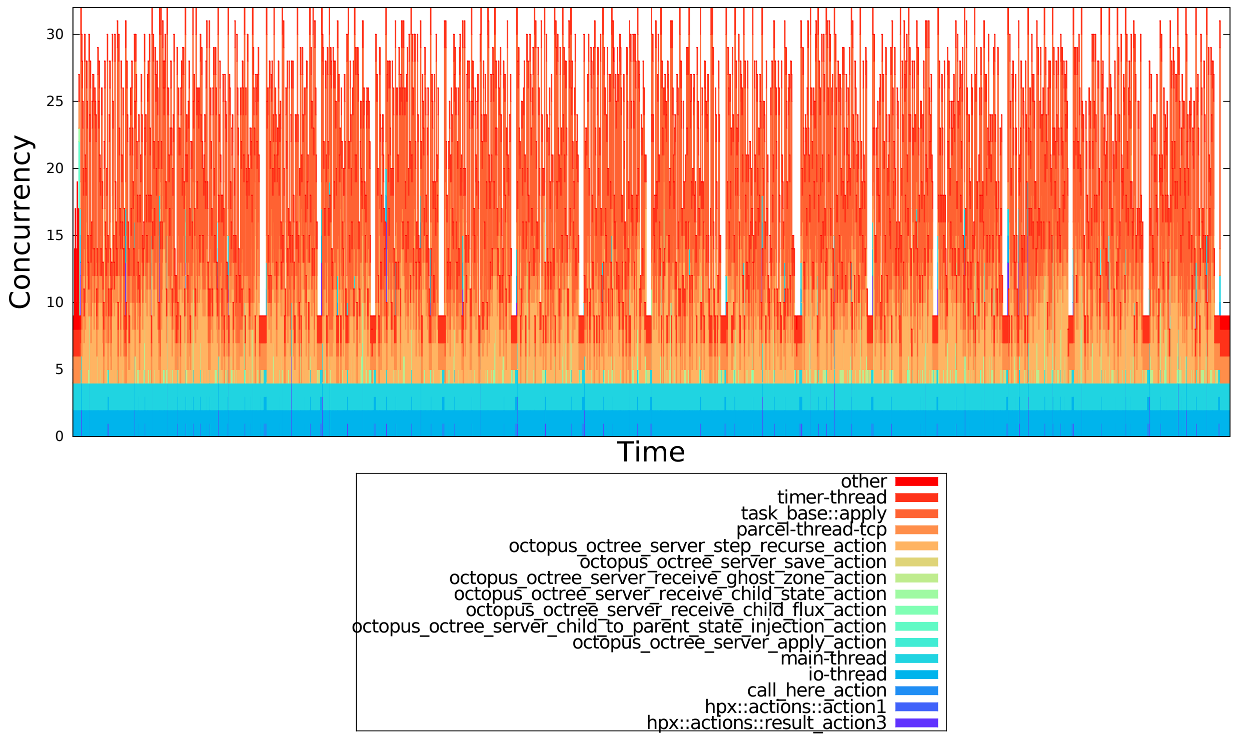
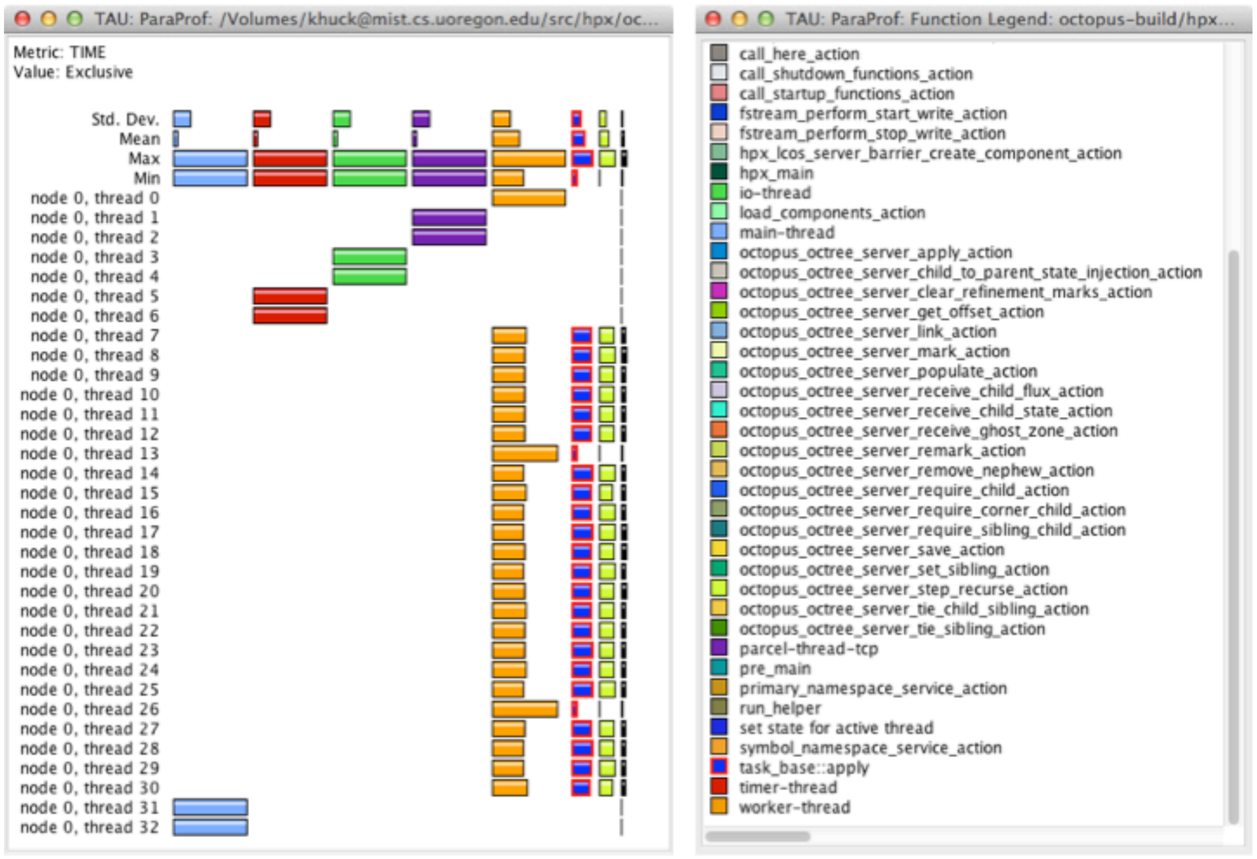


Figure 1: Sampled state of HPX threads, as provided through APEX introspection. The HPX Thread Scheduler reports state changes to APEX through callbacks, and APEX periodically (10 times per second) interrogates the thread state. Dips in concurrency occur during program output and checkpointing at major iteration boundaries. The application is Octopus, running on a 12-core Intel Xeon X5650 2.67GHz system with 24 hardware threads.



🟫 call_here_action
⬜ call_shutdown_functions_action
🟥 call_startup_functions_action
🟦 fstream_perform_start_write_action
⬜ fstream_perform_stop_write_action
🟩 hpx_lcos_server_barrier_create_component_action
⬛ hpx_main
🟩 io-thread
🟩 load_components_action
🟦 main-thread
🟦 octopus_octree_server_apply_action
⬜ octopus_octree_server_child_to_parent_state_injection_action
🟪 octopus_octree_server_clear_refinement_marks_action
🟩 octopus_octree_server_get_offset_action
🟦 octopus_octree_server_link_action
🟨 octopus_octree_server_mark_action
🟩 octopus_octree_server_populate_action
🟪 octopus_octree_server_receive_child_flux_action
🟩 octopus_octree_server_receive_child_state_action
🟧 octopus_octree_server_receive_ghost_zone_action
🟩 octopus_octree_server_remark_action
🟨 octopus_octree_server_remove_nephew_action
🟦 octopus_octree_server_require_child_action
🟩 octopus_octree_server_require_corner_child_action
🟦 octopus_octree_server_require_sibling_child_action
🟨 octopus_octree_server_save_action
🟩 octopus_octree_server_set_sibling_action
🟩 octopus_octree_server_step_recurse_action
🟨 octopus_octree_server_tie_child_sibling_action
🟩 octopus_octree_server_tie_sibling_action
🟪 parcel-thread-tcp
🟦 pre_main
🟫 primary_namespace_service_action
🟫 run_helper
🟦 set state for active thread
🟧 symbol_namespace_service_action
🟥 task_base::apply
🟥 timer-thread
🟧 worker-thread

Figure 2: Profile of Octopus running on a 12-core Intel Xeon X5650 2.67GHz system with 24 HPX worker threads.

Third, we have started defining the performance introspection requirements and architecture. As part of this process, we have started design for the Policy Engine subcomponent. In the proposed design, various components and layers in XSTACK will register policy criteria functions with the Engine. The Policy Engine will be designed as an event listener within APEX, utilizing both the event-driven architecture as well as periodic interrogation. When a policy criteria function evaluates to true (either after an event or during periodic interrogation), a policy action will be taken by APEX, in the form of a callback routine to the component that registered the policy criteria function. In this design, the Policy Engine will be central piece of the introspection puzzle, providing runtime adaptability to the full XSTACK application.

Integration/Introspection

The XPRESS project has a unique opportunity to re-implement the HPC software stack from the application to the OS. Using co-design principles, Xpress is binding previously disjoint parts of the software stack to improve usability and performance. Careful interface design allows the individual layers to be implemented separately, while allowing greater information flow. This information flow allows HPX and LXK to be active participates in an application execution.

Using dynamic performance measures gathered by the hardware, within the OS and Runtime, or by the application itself, RENCI and University of Oregon are developing introspective methods to improve application and system performance. This is done by detecting and reacting to performance problems during execution. Tight integration of the performance tools in HPX and LXK, gives us the ability to impact scheduling decisions at several levels. By reacting to performance bottlenecks in the schedulers, overall system and application performance can be improved.

During a project meeting on interfaces, the fundamental difference between the performance tools, APEX and RCR, was recognized. APEX as a first-person tool is ephemeral and should only live during an application execution. If multiple applications are executing, multiple APEXs exist. RCR as a third-person tool should be long-lived and only one per node is ever active. This recognition facilitated a change in how we think about performance tools and introspection. Rather than being optional tools that are only used when desired, they are integrated parts of HPX and LXK. One result of this is that communication between APEX and RCR is now considered part of the RIOS interface between the runtime and the OS.

UNC/RENCI's focus has been on creating a RCR module that can be added to the OS with minimal impact on the source code base or the execution overheads. This work requires that potions of the RCR framework be made more robust to allow execution by the OS. Placing APEX within HPX has interface implications to UNC/RENCI. The policy engine that uses an energy model to identify introspection opportunities is now within APEX. The interface to RCRblackboard is now part of RIOS and the interface to the HPX thread scheduler is to APEX not RCR directly. This changes some of the technical details of what UNC/RENCI is doing, but should improve overall programmability and performance.

The integration of APEX and RCR is progressing. UNC/RENCI has delivered an updated version of RCR that can run as a standalone daemon on UO systems for testing. The current RCR focuses on bottlenecks associated with memory and identify memory contention on a couple of different architectures. RCR integration with LXK has started and UNC/RENCI is talking to Sandia about the best way to combine the two. RCR can go into LXK or live as a semi-independent module between LXK and the RIOS interface. We expect that decision to be made in the next month. UNC/RENCI is coming up to speed on RIOS, to understand how to best add the APEX/RCR interactions to overall framework.

UNC/RENCI's goals for the remainder of year 2 focus on integration. 1) Determining RCR's exact role with LXK (semi-independent module or embedded component) and providing robust code to create the RCRblackboard within LXK (2) Defining the RIOS extensions for APEX to use RCR information both in the performance tuning modules and its policy engine (3) Add a new thread scheduler to HPX-3 that uses APEX's policy engine (implementation designed to move easily to HPX-4).

LXK

There are three main deliverables for LXK this year. The first deliverable is to deploy an LXK virtual cluster environment that allows for system software development and testing using virtual machines rather than native hardware. The LXK virtual cluster environment has been prototyped and allows several LXK instances running in separate virtual machines to communicate with one another using the Portals 4.0 network API. The motivation for developing the virtual cluster environment was to provide XPRESS collaborators with a more controlled and productive environment for LXK/HPX development compared to running on the native hardware at each site, which is difficult to support due to its diversity. The intended usage model is for a developer to boot several LXK virtual machines in their development workstation to do LXK/HPX development and functional correctness testing. We have not deployed the LXK virtual cluster environment to any collaborator sites yet, but expect to do so in the second half of the year after additional integration work is complete.

The second deliverable is integration of instrumentation capability into LXK. RENCI and Sandia have discussed a plan for integrating RCRtool functionality in the LXK kernel. Currently RCRtool has a user-level daemon that continuously makes system calls to read performance counter hardware registers, which requires OS kernel privilege. Our plan is to re-architect this for LXK such that the LXK kernel does this polling autonomously driven by a kernel-level periodic timer, thus eliminating the system call overhead. LXK will write the performance counter data to a shared memory region that will in turn be mapped into user-level process address spaces. The RIOS layer will provide an interface for controlling the LXK's RCRtool functionality.

The third deliverable is a demonstration of HPX4 running on LXK. Work towards this deliverable within LXK has not started yet, but we expect to be able to complete it by the end of the year. This activity is dependent on progress on HPX-4, which is now sufficient to begin work on the LXK side.

Applications

Application work at Indiana University is proceeding on one proxy application from each of the DOE Co-design centers as well as two other proxy applications led by XPRESS applications team leaders, Mike Heroux and Alice Koniges. Efforts to port the LULESH proxy application (ExMatEx co-design center), NEKBONE (CESAR co-design center), and Exp_CNS_NoSpec (ExaCT co-design center) are currently underway. The LULESH port is now complete and undergoing performance optimization. The GTC particle-in-cell proxy application (NERSC benchmark, PPPL) has already been ported to HPX-3 with results published at SC13 while a port to XPI is currently underway. The HPCG proxy application (http://tiny.cc/hpcg) is currently being ported to XPI by the recommendation from Mike Heroux. Additional application work on the Fast Multipole Method for XPI is being conducted for possible use in particle-in-cell proxy applications as recommended by Alice Koniges. Tom Evans at ORNL is working on SPN "Simplified Spherical Harmonics", which is a mini-app for spherical harmonic capabilities in Denovo. This mini-app is ready to use. A second Monte Carlo transport mini-app is still in development.

Legacy Application Support

Legacy application migration work at University of Houston has completed an initial implementation of the OpenMP programming model on top of HPX-3, named hpxMP. Collaboratively developed with LSU, the hpxMP runtime re-targets the current OpenMP runtime in the OpenUH compiler to the HPX-3 runtime. It supports OpenMP 3.1 features, including parallel region, parallel worksharing, tasking and most synchronization directives and miscellaneous utilities. The implementation has been evaluated using OpenMP validation suites, EPCC OpenMP micro-benchmarks, and NAS parallel benchmarks. We have observed comparable or better performance of the hpxMP to the pthread-version of OpenMP and the Intel OpenMP implementation. Our next step is to perform qualitative studies of performance differences and also to work with LBNL and Sandia to evaluate hpxMP using larger applications recommended by Alice Koniges and Michael Heroux. We will also retarget the hpxMP to the XPI interface when the implementation becomes available.

Researchers at UH have also explored the mechanisms of asynchronous tasking and data-drive computation model across nodes, leveraging MPI+X programming model and the similar work done for the intra-node model. The initial design of extending OpenMP to support overlapping computation and communication in hybrid MPI+OpenMP model is completed, and the implementation is ongoing with plan to be completed around September 2014. With involvement of most industry vendors and national labs, we believe the efforts and results will have profound impacts to the ways we develop large-scale applications.

Miscellaneous Activities and Accomplishments

Representatives from both Sandia and the University of Houston have participated in the OpenMP committee, with Yonghong Yan of UH leading the Interoperability Subcommittee and Stephen Olivier of Sandia leading the Tasking Subcommittee. At the 2013 International Workshop on OpenMP, UH published papers on the OpenMP accelerator model, task dependences, and task profiling. Sandia published a paper on task-generating loops, as well as contributing new material to the OpenMP examples document.

Stephen Olivier of Sandia and Allan Porterfield of RENCI have had a paper accepted at the upcoming 2014 High Performance Power-Aware Computing (HP-PAC) that includes an extension of the work of their HP-PAC 2013 paper on node-level energy-saving dynamic adaptive concurrency to multi-node executions.